APPENDIX

Values shown in the #define statements are for illustation only.

Function "InitializeBuffer" sets all History Buffer elements to zero upon initialization. This is required only if the output is sampled before the first L elements are entered.

```
#define     L       1070        // Number of elements in the History Buffer
#define     M       107         // Number of M elements in the History Buffer

#define     TH_HB_1A  6         // TH_HB for network 1A
#define     TH_HB_1B  20        // TH_HB for network 1B
#define     TH_HB_2A  7         // TH_HB for network 2A
#define     TH_HB_2B  25        // TH_HB for network 2B

#define     TH_PR_1A  300       // TH_PR for network 1A
#define     TH_PR_1B  300       // TH_PR for network 1B
#define     TH_PR_2A  30        // TH_PR for network 2A
#define     TH_PR_2B  30        // TH_PR for network 2B

int         History_Buffer[L];// The History Buffer of L elements


void    InitializeBuffer(void)
{
        // Set all elements to zero
        int i;

        for (i=0 ; i<L ; i++)
                History_Buffer[i] = 0;
}


int ProcessNewHistoryBufferElement(int NewValue)
{
        // Illustrates placing one new observation into the History Buffer
        // then computing the resulting system state.

        // Input:       NewValue - number of rewritten sub-units for this Data Set
        // Output:      State - the resulting system state
```

```
// For clarity, this illustration assumes that input elements are
// placed in the last element of the History Buffer.  Doing so requires
// that the entire History Buffer be shifted with each new entry.
// An efficient implementation would use pointers to maintain input
// and output indices eliminating the need for shifting the buffer.

// The origin is zero (counts start at zero).  Therefore the first
// location in the History Buffer is History_Buffer[0] and the last
// location is History_Buffer[L-1].

int     counter;
int     i;
int     Pa, Pb, Qa, Qb;
int     State;


// Rotate History Buffer: discarding oldest entry, making room for new
for (i=1 ; i<L ; i++)
        History_Buffer[i-1] = History_Buffer[i];

// Place new element into History Buffer
History_Buffer[L-1] = NewValue;


// EVALUATE OBSERVER NETWORK #1A over L elements
// Count number of History Buffer items that equal or exceed TH_HB_1A

counter = 0;

for (i=0 ; i<L ; i++)
        if ( TH_HB_1A <= History_Buffer[i])
                counter = counter + 1;

// Set Pa = 1 if the count equals or exceeds TH_PR_1A
if ( TH_PR_1A <= counter)
        Pa = 1;
else
        Pa = 0;



// EVALUATE OBSERVER NETWORK #1B over L elements
// Count number of History Buffer items that equal or exceed TH_HB_1B

counter = 0;
```

```
for (i=0 ; i<L ; i++)
        if ( TH_HB_1B <= History_Buffer[i])
                counter = counter + 1;

// Set Pb = 1 if the count equals or exceeds TH_PR_1B
if ( TH_PR_1B <= counter)
        Pb = 1;
else
        Pb = 0;


// EVALUATE OBSERVER NETWORK #2A over M elements
// Count number of History Buffer items that equal or exceed TH_HB_2A

counter = 0;

for (i=L-M ; i<L ; i++)
        if ( TH_HB_2A <= History_Buffer[i])
                counter = counter + 1;

// Set Qa = 1 if the count equals or exceeds TH_PR_2A
if ( TH_PR_2A <= counter)
        Qa = 1;
else
        Qa = 0;



// EVALUATE OBSERVER NETWORK #2B over M elements
// Count number of History Buffer items that equal or exceed TH_HB_2B

counter = 0;

for (i=L-M ; i<L ; i++)
        if ( TH_HB_2B <= History_Buffer[i])
                counter = counter + 1;

// Set Qb true if the count equals or exceeds TH_PR_2B
if ( TH_PR_2B <= counter)
        Qb = 1;
else
        Qb = 0;


// Combine the Pa, Pb, Qa, and Qb in a convenient manner for output
```

```
        // For illustration, convert them to value 0 through 15

        State = Qb + 2*Pb + 4*Qa + 8*Pa;

        return State;
}
```